MASSACHUSETTS INITITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 89-565
October 1989

A Locality-Based Multiprocessor Cache Interference Model

Anant Agarwal

Abstract

Keeping data consistent in cache-coherent multiprocessors often requires the invalidation of cached blocks and results in higher miss rates. The increase in the cache miss rate due to invalidations is related to the number of processors and to the level of sharing. Analytically modeling this increase is important for the accurate performance evaluation of cache-based multiprocessors. Previous modeling efforts assumed a uniform probability of access to shared blocks. However, applications that we have studied show significant temporal locality of access to shared blocks, which substantially alters the magnitude of sharing related misses. This paper develops a multiprocessor cache interference model using temporal locality information measured from address traces of parallel applications. The model is very simple and yields reasonable predictions.

Microsystems
Technology
Laboratories

Massachusntts
Institute
of Technology

Cambridge
Massachusetts
02139

Room 39-321
Telephone
(617) 253-0292

Acknowledgements

This research was supported by the Defense Advanced Research Projects Agency under contract N00014-87-K-0825, and by grants from the Sloan Foundation and IBM.

Author Information

Agarwal: Laboratory for Computer Science, Room NE43-418, MIT, Cambridge, MA 02139. (617) 253-1448.

A Locality-Based Multiprocessor Cache Interference Model

Anant Agarwal
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

Keeping data consistent in cache-coherent multiprocessors often requires the invalidation of cached blocks and results in higher miss rates. The increase in the cache miss rate due to invalidations is related to the number of processors and to the level of sharing. Analytically modeling this increase is important for the accurate performance evaluation of cache-based multiprocessors. Previous modeling efforts assumed a uniform probability of access to shared blocks. However, applications that we have studied show significant temporal locality of access to shared blocks, which substantially alters the magnitude of sharing related misses. This paper develops a multiprocessor cache interference model using temporal locality information measured from address traces of parallel applications. The model is very simple and yields reasonable predictions. K[illegible]nds)

Index terms: Cache coherence, Shared-memory multiprocessors, processor locality, cache interference, cache miss rate, multiprocessor performance evaluation. (P[illegible]

1 Introduction

In an invalidation-based cache coherence scheme, when a processor writes into a block currently cached by one or more other processors, each of the other cached copies must be purged. These invalidations increase the cache miss rate. Invalidation-based cache coherence schemes include several snoopy-cache protocols [1, 2, 3] and the directory-based protocols [4, 5, 6, 7]. Most commercial bus-based multiprocessors also use similar schemes for cache coherence.

The cache miss rate of an individual processor depends on the level of sharing, the number of processors, and the characteristics of program accesses to shared memory. Dubois and Briggs [8] estimated the miss rate of a cache in a multiprocessor environment assuming that every shared block was equally likely to be accessed. The miss rates obtained turn out to be very pessimistic for the applications we measured. The chief reason for the discrepancy was that references to shared memory displayed temporal locality in much the same manner private references did. We will show how temporal locality information can be extracted from parallel address traces and used in a simple multiprocessor interference model to get good estimates of multiprocessor cache miss rates.

Several recent studies addressed the issue of locality in multiprocessor memory referencing. Agarwal and Gupta [9] proposed using the notion of processor locality, which is measured as the number of local accesses of a block of shared data before a remote write. They also studied the time intervals between references of local and remote processors as a measure of temporal locality. Eggers and Katz [10] used the lengths of write runs to characterize multiple local writes to a shared block before a non-local reference. Dubois and Wang [11] used the lengths of bursts of local processor references to shared blocks as a measure of locality.

Both the analytical modeling study of Dubois and Briggs and the simulation model of Archibald and Baer assumed that each shared block was equally likely to be accessed during any cycle as in the Independent Reference Model [12]. The Dubois and Briggs model was quite complex and built the sharing related invalidation model on top of a single processor cache model assuming the LRU stack distribution for the private references. Dubois and Wang [11] recently extended this work to include the burst access notion of locality. Bursts were defined as a sequence of uninterrupted local accesses. The burst lengths and number of bursts were measured from parallel applications. Because the burst length changes with the number of processors in the system, separate measurements must be carried out for each number of processors. Although in some applications the bursts might occur over a small enough interval that the effective length remains the same even if the number of processors is increased. Our focus here is to obtain a simple model for the miss rate as a function of the number of processors. Our model uses the time interval between successive references to a block by a given processor – a parameter independent of the number of processors in the system – to characterize locality.

2 The Model

As in the Dubois and Briggs study, we will denote the cache under consideration the local cache and all the others as remote. We will assume an infinite cache. Several recent multiprocessor studies have made this assumption. Finite cache effects can be easily factored in using single processor cache models. Furthermore, by focusing on the invalidation misses alone we can simplify the model greatly. In practice, network traffic will also result from writebacks and local writes to clean objects. However, the focus of this paper is on misses which necessarily stall the processor. Both writebacks and local writes to clean objects (assuming buffered invalidations and a weak coherence model) can proceed without stalling the processor.

A shared block can have two cache states: present (*Pres*) or absent (*Abs*). In an infinite cache, the mechanism that causes a block to change state from present to absent is the invalidation signal initiated by a remote write reference, while a local reference to the block ensures its future presence. Let the probability of a reference to a shared block during any cycle be s, the probability of a write be w, and the number of processors in the system be P. Instruction blocks are treated as private as they are never written. Given a shared reference, let the probability of a request to a specific shared block be p_r, and let p_l be the corresponding probability of an access from the local processor.

The probability of an invalidation in a cycle from any remote processor to a given shared block is swp_r, which is simply the product of the probabilities of accessing a shared block, the write probability, and the probability of a remote reference to the given block. If there are P processors, assuming the accesses from different processors are independent, the probability that no remote processor writes into the shared block in any cycle is $(1 - swp_r)^{P-1}$. Then, the probability of an invalidation is one minus the probability no processor writes to the block during that cycle. The invalidation probability during a cycle is given by

$$1 - (1 - swp_r)^{P-1}$$

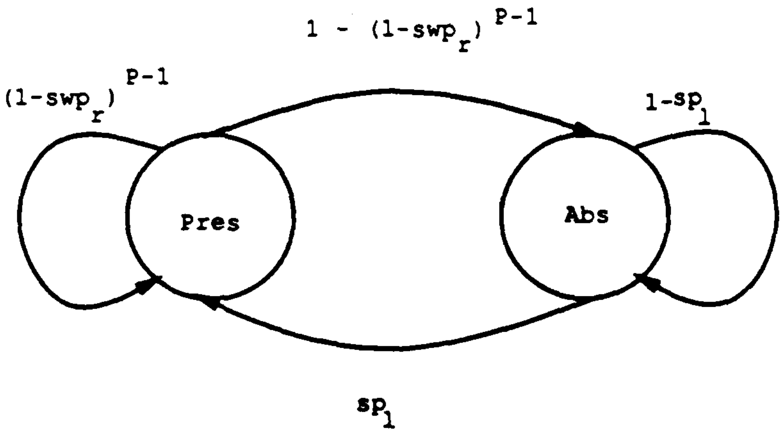

Figure 1: States of a cache block.

If the block is absent from the local cache, the probability it returns to the cache on an access is simply the probability of a local reference to that shared block, which is sp_l. These transition probabilities can be represented in a state transition diagram as in Figure 1.

The steady-state probability the block is absent from the cache is

$$p_{abs} = \frac{1-(1-swp_r)^{P-1}}{1-(1-swp_r)^{P-1}+sp_l}$$

In an infinite cache, the probability of a miss (p_m) is the probability of a shared reference times the probability the shared block is absent, and is given by,

$$p_m = s\,\frac{1-(1-swp_r)^{P-1}}{1-(1-swp_r)^{P-1}+sp_l} \tag{1}$$

If the probability of a write from a remote processor to a given block in any cycle is much smaller than one, that is, if swp_r is small, we can simplify the above equation to get,

$$p_m \approx \frac{s}{1+\frac{p_l}{wp_r(P-1)}} \tag{2}$$

When $wp_r(P-1)$ is small compared to p_l, which happens for high temporal locality of local accesses, we can further simplify the miss rate expression as,

$$p_m \approx \frac{swp_r(P-1)}{p_l} \tag{3}$$

2.1 The IRM Model for Shared References

Let the number of shared blocks be B. Using the Independent Reference Model for the shared blocks, each shared block is equally likely to be accessed on a shared reference. Then, $p_l = p_r = 1/B$. This yields a model similar to that used by Dubois and Briggs.

$$p_m = s\frac{1-(1-sw/B)^{P-1}}{1-(1-sw/B)^{P-1}+s/B} \quad (4)$$

2.2 The Locality-Based Model

In the model discussed in the previous section, the probability of a remote processor accessing a given shared block is equal to the probability the local processor accesses it. We have observed that repeat local accesses of a given block are more likely than writes from remote processors [9]. Several common programming practices encourage such locality. Often, modifications of a set of shared blocks are performed within critical sections that preclude accesses from the other processors. In some systems, chunks of shared work to be performed are placed on queues and assigned to idle processors. The set of block accessed by a processor is then related to the chunk of work it has been assigned. Such locality properties in accessing shared memory is not only a feature of current parallel programming techniques, but should be encouraged as much as possible to reduce cache contention.

We will now derive an interference model based on slightly different assumptions than Dubois and Briggs. We will assume as before that the probability of a remote reference to a shared block (p_r) is uniform for all the blocks and is equal to $1/B$ as before. The probability a shared reference from the local processor is to the given block (p_l) will be greater than p_r if locality holds. The locality parameter p_l depends on the shared memory reference characteristics of the parallel application, and is measured as the reciprocal of the average time interval between repeat references to a shared block by a given processor. To compute p_r we measure the number of shared blocks, the reciprocal of which is p_r.[1] s is measured as the fraction of shared references. and w is the fraction of writes. The miss rate can now be written as,

$$p_m = s\,\frac{1-(1-sw/B)^{P-1}}{1-(1-sw/B)^{P-1}+sp_l} \quad (5)$$

3 Address Traces

The model predictions are compared with simulations using three parallel address traces. The traces used are POPS, P-Thor and LocusRoute. POPS [13] is a parallel implementation of a rule-based programming language called OPS5, P-THOR is a parallel logic simulator implemented by Larry Soule, and LocusRoute is a parallel VLSI router written by Jonathan Rose at Stanford [14]. Each of these applications has the characteristic that the processes obtain pieces of work to perform from a large set.

The four-processor POPS and P-THOR traces were obtained using a multiprocessor extension of the ATUM tracing scheme [15]. ATUM stands for Address Tracing Using Microcode. In ATUM the microcode writes the addresses touched by the program to a reserved portion of memory as a side effect of normal execution. In the multiprocessor extension of ATUM, each access to trace memory is interlocked to enable the microcode in several processors to write their

[1] In applications where the remote shared references favor the blocks being accessed by the local processor, the probability of a remote access p_r can be measured from the trace using the reciprocal of the average interval between remote references by a given processor to the shared block. Note that this average is also unrelated to the number of processors in the system.

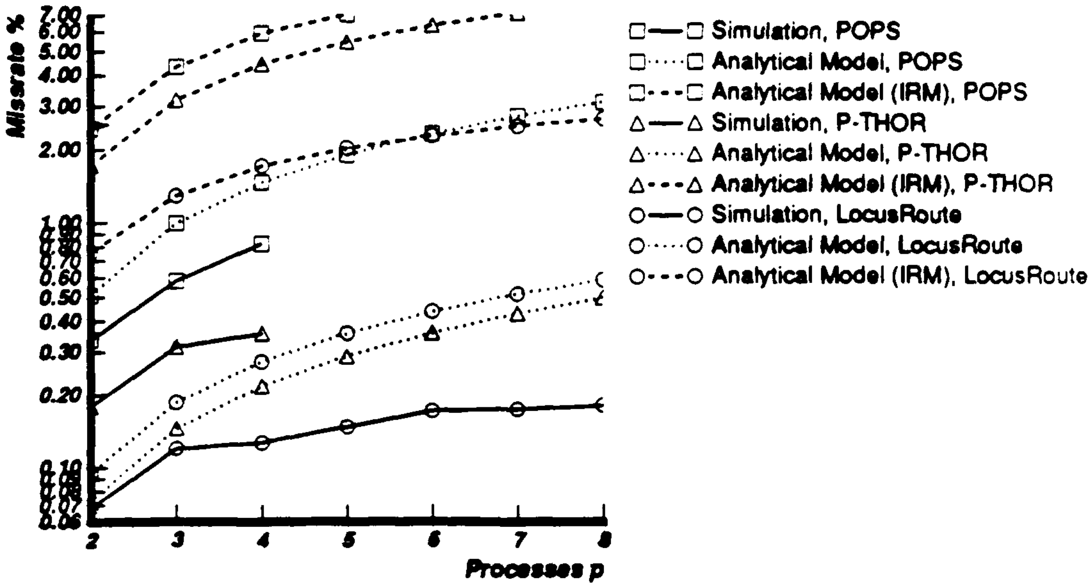

Figure 2: Invalidation induced cache miss rate.

references to this memory. Thus a trace contains interleaved address streams of several processors. The traces used for this study were gathered on a 4-CPU VAX 8350 machine running the MACH operating system. ATUM also records virtual to physical translation information that allows a postprocessor to create a physical address trace required to detect sharing. Each trace :s roughly 3.5 million references long. For details regarding how time is counted in these traces and trace details the reader is referred to [9].

The eight-processor LocusRoute is obtained from a parallel tracer implemented by Steve Goldschmidt at Stanford that used the VAX T-bit. The basic idea behind multiprocessor T-bit tracing is to schedule a new process on every trap instruction. After a process suffers a trap, and the corresponding memory address is recorded, the scheduler saves the processor state of the trapped process and schedules another process from its list of processes, typically in a round-robin fashion. This method yields round-robin interleaved address traces of multiple processes, where all addresses corresponding to a given process occur at fixed intervals in the trace.

4 Results and Discussion

Figure 4 plots the predictions of the locality based model (using the accurate expression), and those using the Independent Reference Model, and compares these with simulations of the three parallel address traces. Figure 4 shows the graphs averaged over the three applications. The models are driven using parameters measured from the traces.

The figures show that the model performs reasonable well in predicting the cache miss rate component due to invalidations. Without considering locality, the miss rate is overestimated by more a factor of 10. Put another way, if parallel applications are carefully written to enhance locality of access to shared blocks, substantial performance benefits can arise.

How does block size impact the invalidation rate? The multiprocessor interference component of misses displays a simple dependence on s and the local and remote locality parameters, p_l and

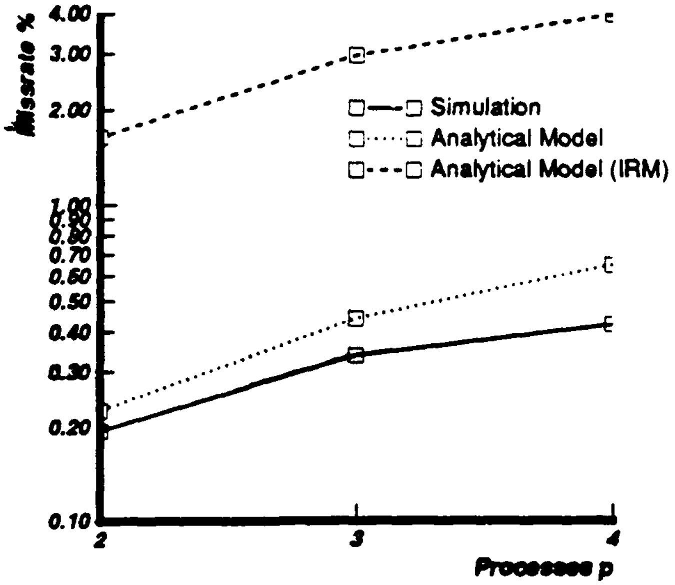

Figure 3: Invalidation induced average cache miss rate.

p_r, as shown in Equation 3. The effect of block size on the invalidation rate is then related to how the fraction of shared references, s, and these parameters vary with block size. If a larger block places words used by the same processor into one cache block, p_l increases yielding a lower miss rate, while the opposite is true if unrelated words referenced by remote processors are placed in the same block. Several recent studies have observed that increasing block size can hurt or improve performance depending on the application's memory referencing behavior [16, 17].

In this paper, we considered invalidating protocols. Some cache coherence protocols update remote copies with the new value [18, 19] and do not hurt the miss rate. However, the write update rate will depend on the above factors, and we believe similar ideas can be used in deriving update rates for such protocols.

The model we derived leads to a very simple approximation for the cache interference in multiprocessors. It shows that the coherence related miss rate increases linearly with the number of processors. The model can be used in multiprocessor performance evaluation. Its simplicity encourages its use in software systems, such as in schedulers. A scheduler must address the tradeoff between locality (and hence lower miss rates) and load balancing. Models of this nature can be used to make such tradeoffs. We are investigating an affinity-based scheduler that includes this model.

In parallel applications, commonly, most blocks are not used by all processors. For example, in some physical problems with near neighbor communication, a block might be shared among a few of the processors. In such cases, the model can be modified so that the miss rate has a bilinear form; the miss rate initially follows the usual linear model until the maximum number of processors sharing the block is reached, and then flattens out. Such effects have been noticed by several studies [11, 20, 21].

5 Conclusions

This paper presented a temporal locality based multiprocessor interference model that predicts the cache miss rate as the number of processors in the system increases. Comparison with simulations against parallel applications demonstrates that the model predicts the interference-related miss component reasonably well. When the probability of invalidation from remote processors is small, the miss rate bears a simple linear relationship to the number of processors in the system. The predictions of a model using no locality information are very pessimistic. More importantly, we see that the miss rate of a coherent cache for an application with significant temporal locality is much lower than when such locality is absent.

6 Acknowledgments

Discussions with Susan Owicki are gratefully acknowledged. The research reported in this paper is funded by DARPA contract # N00014-87-K-0825, and by grants from the Sloan foundation and IBM.

References

[1] R. H. Katz, S. J. Eggers, D. A. Wood, C. L. Perkins, and R. G. Sheldon. Implementing a Cache Consistency Protocol. In *Proceedings of the 12th International Symposium on Computer Architecture*, pages 276–283, IEEE, New York, June 1985.

[2] James R. Goodman. Using Cache Memory to Reduce Processor-Memory Traffic. In *Proceedings of the 10th Annual Symposium on Computer Architecture*, pages 124–131, IEEE, New York, June 1983.

[3] Mark S. Papamarcos and Janak H. patel. A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories. In *Proceedings of the 12th International Symposium on Computer Architecture*, pages 348–354, IEEE, New York, June 1985.

[4] Peiyi Tang, Pen-Chung Yew, and Chuan-Qi Zhu. *Processor Self-scheduling in large Multiprocessor Systems.* Technical Report Report No. 536, Center for Supercomputing Research and Development, U. of Illinois at Urbana-Champaign, October 1985.

[5] Lucien M. Censier and Paul Feautrier. A New Solution to Coherence Problems in Multicache Systems. *IEEE Transactions on Computers*, C-27(12):1112–1118, December 1978.

[6] James Archibald and Jean-Loup Baer. An Economical Solution to the Cache Coherence Problem. In *Proceedings of the 12th International Symposium on Computer Architecture*, pages 355–362, IEEE, New York, June 1985.

[7] Anant Agarwal, Richard Simoni, John Hennessy, and Mark Horowitz. An Evaluation of Directory Schemes for Cache Coherence. In *Proceedings of the 15th International Symposium on Computer Architecture*, IEEE, New York, June 1988.

[8] Michel Dubois and Faye A. Briggs. Effects of Cache Coherence in Multiprocessors. In *Proceedings of the 9th International Symposium on Computer Architecture*, pages 299–308, IEEE, New York, May 1982.

[9] Anant Agarwal and Anoop Gupta. Memory-Reference Characteristics of Multiprocessor Applications under MACH. In *Proceedings of ACM SIGMETRICS 1988*, May 1988.

[10] S. J. Eggers and R. H. Katz. A Characterization of Sharing in Parallel Programs and Its Application to Coherency Protocol Evaluation. In *Proceedings of the 15th International Symposium on Computer Architecture*, IEEE, New York, June 1988.

[11] Michel Dubois and Jin-Chin Wang. Shared Data Contention in a Cache Coherence Protocol. In *Proceedings ICPP*, August 1988.

[12] J. R. Spirn. *Program Behavior: Models and Measurements. Operating and Programming Systems Series*, Elsevier, New York, 1977.

[13] Anoop Gupta, Charles Forgy, and Robert Wedig. Parallel Architectures and Algorithms for Rule-Based Systems. In *Proceedings of the 13th Annual Symposium on Computer Architecture*, IEEE, New York, June 1986.

[14] Jonathan Rose. LocusRoute: A Parallel Global Router for Standard Cells. In *Design Automation Conference*, pages 189–195, June 1988.

[15] Richard L. Sites and Anant Agarwal. Multiprocessor Cache Analysis using ATUM. In *Proceedings of the 15th International Symposium on Computer Architecture*, pages 186–195, IEEE, New York, June 1988.

[16] Anant Agarwal and Anoop Gupta. *Temporal, Processor, and Spatial Locality in Multiprocessor Memory references.* Technical Report, MIT VLSI Memo, April 1989. Submitted for publication.

[17] Susan J. Eggers and Randy H. Katz. The Effect of Sharing on the Cache and Bus Performance of Parallel Programs. In *Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)*, April 1989.

[18] E. McCreight. *The Dragon Computer System: An Early Overview.* Technical Report, Xerox Corp., September 1984.

[19] Charles P. Thacker and Lawrence C. Stewart. Firefly: a Multiprocessor Workstation. In *Proceedings of ASPLOS II*, pages 164–172, October 1987.

[20] Wolf-Dietrich Weber and Anoop Gupta. Analysis of Cache Invalidation Patterns in Multiprocessors. In *Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)*, April 1989.

[21] Anant Agarwal and Mathews Cherian. Adaptive Backoff Synchronization Techniques. In *Proceedings 16th Annual International Symposium on Computer Architecture*, IEEE, New York, June 1989.